Let's try a different analogy: **Finding a specific page in a book** versus **Finding a "Buggy" version of an app**.

---

1. Finding a Specific Page (while (left <= right))

Imagine you are looking for **Page 42** in a textbook.

- **Process:** You open the book to a random page.
- **Action:** * If you see Page 42, **you stop.** You found it!
  - If you see Page 50, you know Page 42 must be in the range $[1, 49]$.
  - If you see Page 30, you know Page 42 must be in the range $[31, 100]$.
- **The "Last Stand":** Eventually, you might be down to just **one page** left to flip. You still need to look at that page to see if it's Page 42 or if the page is missing entirely.
- **Why <=:** You need the loop to run even when there is only **one** candidate left.

---

2. Finding the "First Broken Version" (while (left < right))

Imagine you are a developer at WhatsApp. A new update caused a bug. You know **Version 1** was fine, but **Version 100** has the bug. You want to find the **earliest** version that broke.

- **Process:** You check Version 50.
- **Action:** * If Version 50 is **broken**, it might be the *first* broken one, or the break might have happened earlier (like Version 40). So, you keep 50 in your search range. (right = mid)
  - If Version 50 is **working**, you know for a fact the bug started at Version 51 or later. You discard 50. (left = mid + 1)
- **The "Convergence":** You aren't looking for a specific number like "42." You are narrowing down the "Bad Zone" until it shrinks to a single point.
- **Why <:** Once left and right meet at the same version, there is no "middle" left to check. That meeting point **is** your answer. If you used <=, the loop would keep running forever because left would never get larger than right.

Summary of the "Vibe"

Type	The "Searcher" (<=)	The "Squeezer" (<)
Goal	"Is this the one?"	"Where is the transition?"
Action	Jump out as soon as you find it.	Keep squeezing until you can't anymore.
Result	Success (index) or Failure (-1).	The location of the change.

A Quick Rule of Thumb

If your code has an if (arr[mid] == target) return mid; inside the loop, you almost always want **while (left <= right)**.
If you are trying to find a "Minimum" or "First X that satisfies Y," and you don't have a return inside the loop, you want **while (left < right)**.